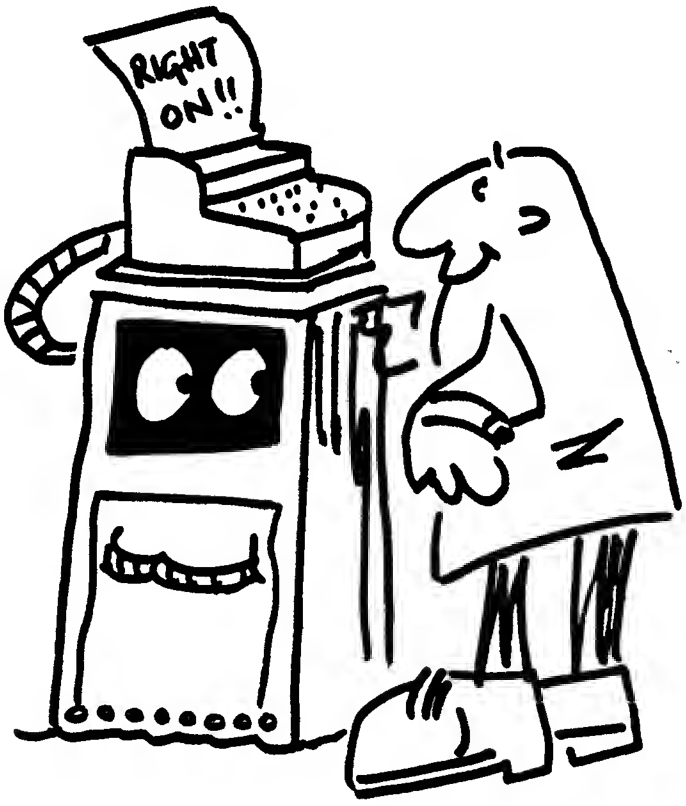
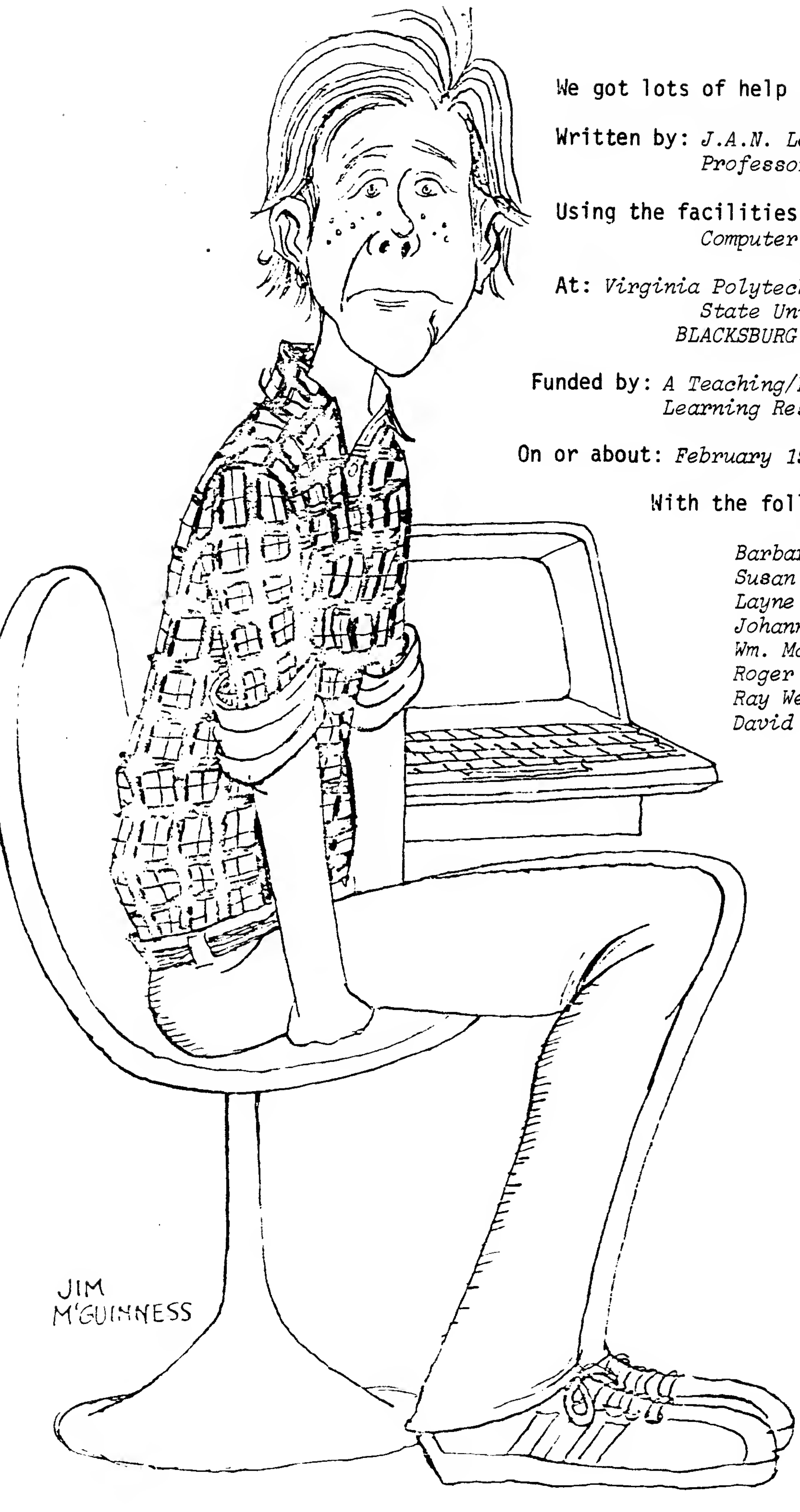
so! you want to use Multics* ?



WELL THE FIRST JOB IS TO FIND
A VACANT TERMINAL
(WE STRONGLY SUGGEST LOOKING
IN McBRYDE 116 OR
ROBESON 120)

WRITTEN AND DRAWN AT VIRGINIA TECH FOR VIRGINIA TECH STUDENTS AND FACULTY
BY J.A.N.LEE, DEPARTMENT OF COMPUTER SCIENCE, FEBRUARY 1979

* *MULTiplexed Interactive Computing Service*



We got lots of help putting this together:

Written by: *J.A.N. Lee*
Professor of Computer Science

Using the facilities of: *The Department of Computer Science*

At: *Virginia Polytechnic Institute and State University*
BLACKSBURG VA 24061

Funded by: *A Teaching/Learning Grant from the Learning Resources Center*

On or about: *February 1979*

With the following reviewers:

Barbara Love
Susan Bright
Layne Watson
Johannes Martin
Wm. McCormack
Roger Ehrich
Ray West
David Roper

Picture Credits:

Datamation
Creative Computing
Brooks/Cole Pub. Co.
Peoples Computer Co.

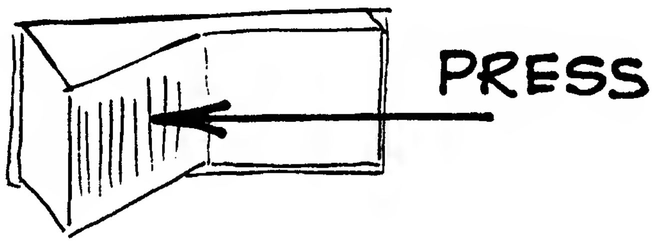
LETS

ASSUME

(USUALLY A VERY BAD THING TO DO - ESPECIALLY IN PROGRAMMING)

THAT YOU HAVE FOUND A TERMINAL AND
IT SAYS "HEWLETT-PACKARD" SOMEWHERE ON IT
(IF NOT GO FIND SOMEONE WHO CAN HELP, WE CAN'T)

reach around the back on the left hand side
and find a rocker switch - turn it on.



AFTER A WHILE SOMETHING LIKE THE FOLLOWING WILL APPEAR ON THE SCREEN:

```
GO HOKIES - - - STOMP UNIVERSITY OF VIRGINIA
Multics MR 6.5h: Virginia Tech Computing Center
Load = 23.0 out of 72.0 units: Users = 23
```

THEN

NOTHING

SO

PANIC

NOT REALLY

the Multics system told you who it was
NOW it is waiting for you to introduce yourself
BUT unfortunately it will only respond to you
IF the Computing Center has introduced you two
<u>PREVIOUSLY</u>

WHAT YOU NEED IS A

person_id

(Computers use an underline in the same way as you might use a hyphen because they think a hyphen is a minus sign and that doesn't make much sense to them !!!)

To get a person_id you need to see your faculty adviser or your course instructor and get a Computing Center Form entitled:

"Multics System Person Registration"

Fill it out (or in) carefully - your person_id will be your name and initials

A **password** is something else you need !

One of the questions on the registration form was to fill in your birthdate - that wasn't so that the computer could send you a card ! Since most people don't know your birthdate, the computer initially uses that as your own special password - it is four digits - the birth month (two digits, first one zero if necessary) and birth day (also two digits) without a space in between.

If you were born on July 4th, then your password would be 0704.

Later on we'll tell you how to change your password to (almost) anything so that even the Computing Center won't know it!

project_id

NOT ONLY MUST YOU HAVE AN INTRODUCTION TO THE COMPUTER BUT YOU MUST ALSO BE ACKNOWLEDGED BY SOMEONE'S PROJECT

© DATAMATION ®

THAT SOMEONE IS THE SOMEONE WHO IS GOING TO PAY FOR YOUR COMPUTER TIME !!!!

YOUR ADVISER OR COURSE INSTRUCTOR WILL TELL YOU WHAT YOUR project_id IS TO BE .... IT WILL PROBABLY BE SOMETHING LIKE CS3378W82

WHICH IS SUPPOSED TO MEAN COURSE CS3378 IN WINTER QUARTER 1982

TYPING

The keys on the terminals have lots of characters engraved on them, but which should you use - or better WHICH DOES THE COMPUTER USE?

If there are symbols on the front of the keys as well as on top, then the symbols on the front are the ones.

Multics uses both upper case (shift) symbols as well as lower case, your person_id will usually have an upper case first letter in your last name and upper case initials.

If your name is Uncle Sam (first name Uncle, last name Sam, no middle name) then your person_id would be

SamU

watch out

for the differences between 1(One) and l(ell),
between Ø(zero) and O(oh),
between (space) and _(underline),
and some others - look carefully at the keys.

MISTRAKES

#

If you make a typing error, you can tell the computer to ignore the immediately previous character by typing #.
So *mistr#akes* would actually be communicated to the computer as *mistakes*. If you make several errors, or if you notice the error several characters back, you can erase the characters by one # for each character to be removed.
SO... *participation####nt* would really be *participant* .

(Note that consecutive spaces count as one character)

@

IF YOU MAKE SO MANY TYPING MISTAKES IN ONE LINE,
or the mistake is so far back that you can't count the
number of # that you need ...
THEN type @. It wipes out the whole line up to that
point and you can start again.

SO *this is a real mess@this is OK* comes out as *this is OK*

my computer likes me when I hit

RETURN

Just in case you haven't done all the correcting of mistakes
the computer waits to act on your instructions until you finish
the line with a carriage return (usually just marked "RETURN")

SO just before you finish the line with a return - check your typing!!!

ARE YOU READY NOW - OK, GO AND FIND AN UNUSED TERMINAL AND LET'S START AGAIN.

TURN ON THE TERMINAL AGAIN (REMEMBER ???)

after it has done its thing in telling you who it is type in:

```
login SamU CS3378W82 (return) *
```

The computer will then respond with

```
Password:
```

This is where you are supposed to type in your personal, secret, password that you have. At the beginning that is your birthdate - month and day. So Uncle Sam would enter:

```
0704 (return)
```

W H O O P S NOTHING PRINTED OUT RIGHT ? ? ?

OK DON'T PANIC - IT'S NOT SUPPOSED TO PRINT ANYTHING

Your password is so secret that the computer doesn't even print it out in case someone is looking over your shoulder, so be very careful in typing it in. It is very difficult to correct mistakes when you cannot see what you typed.

If anything has gone wrong to this point, the system will tell you what to do next. If you really have problems, first try typing in:

```
help login (return)
```

and the computer will tell you more about how to login.

If you have been accepted into the system, then you will be told about whether you are to be pre-empted or not. That is an indication of how long you have before someone else gets a fair share of the computer.

* This is the last page on which we are going to remind you to finish each line with a carriage return.

before you runn# you gotta WRITE

You probably got on to the system to run some programs, right? If not skip this section.

If you are going to use someone else's program then you can skip this section - ask them how to run their program.

EDITING

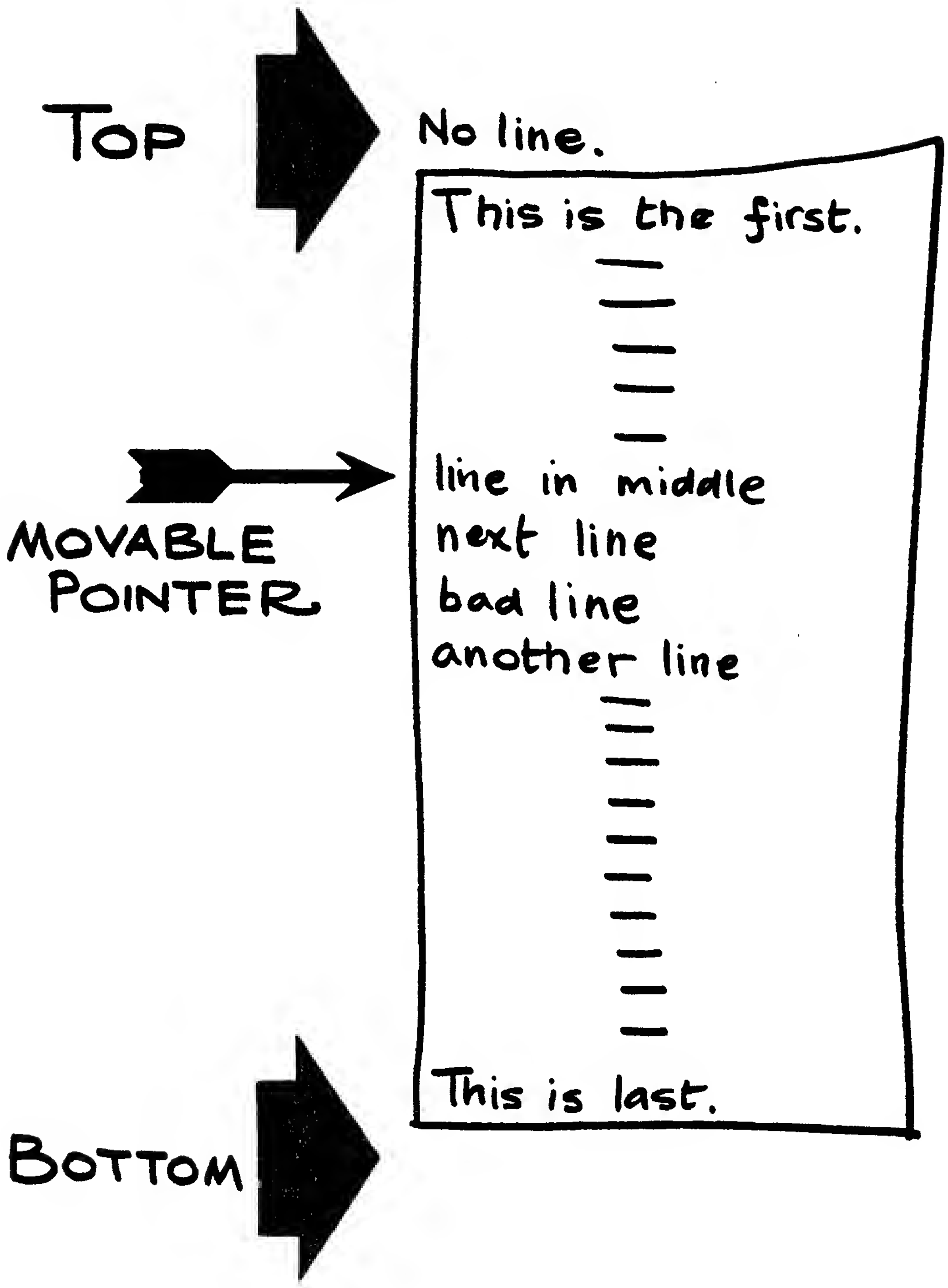
In order to tell the computer to do something, you first have to write a program to do it. And that program has to be given to the machine. So first we have to transcribe the program from hand written notes to something the computer can read.

Since this is very much like the job that a reporter does for a newspaper, we call the job "EDITING" and the computer system which does it is named *edm* .

Instead of producing a page of a paper, we produce something called a *segment*.

And since most programmers have lots of these around, we give each segment a title or name - the segment name.

parts of a SEGMENT



The edm editor system workes in one of two modes, just like a typist !

INPUT

In the input mode, anything you type in gets stored by the computer in the segment which you are constructing. It saves your information by lines, each line being recognized by your use of the return key.

THE FIRST TIME YOU USE THE EDITOR
FOR A NEW SEGMENT, THE COMPUTER
WILL AUTOMATICALLY ASSUME YOU
WANT TO BE IN THE INPUT MODE.

You can recognize the input mode because the system types *INPUT:*

EDIT

The other thing that a reporter does is to correct all his mistakes. If you catch them before you hit return, then you can use # and @ to make corrections. Afterwards you have to do some serious editing.

THE EDIT MODE LETS YOU PRINT OUT CLEAN
COPIES OF THE SEGMENT, LOOK AT INDIVIDUAL
LINES AND MAKE CORRECTIONS.

changing mode:

If you are in one mode (input or edit) you can change to the other mode by typing a line which contains <u>ONLY</u> a period. The new mode will then announce itself. So if you forget what mode you are in - just type . and the <u>other</u> will appear - BUT DON'T FORGET TO GO BACK!!!

INPUT

Lets assume that you are going to input the handwritten program below, which is suppoed##sed to be in the PL/1 language.

```
mortgage: procedure;
  declare (c,p,r) float decimal,
        (n,i) fixed decimal;
      get list (c,p,r,n);
       i = 1;
       do while (i <= n);
          c = c * (1.0e0 + i * 0.01e0) - r;
          put skip list ('after year', i, 'balance is', c);
          i = i+1;
          end;
    end mortgage;
```

Lets call this program *program1*. So, to get the editor to work for us <u>and</u> to create a new segment named *program1* we enter:

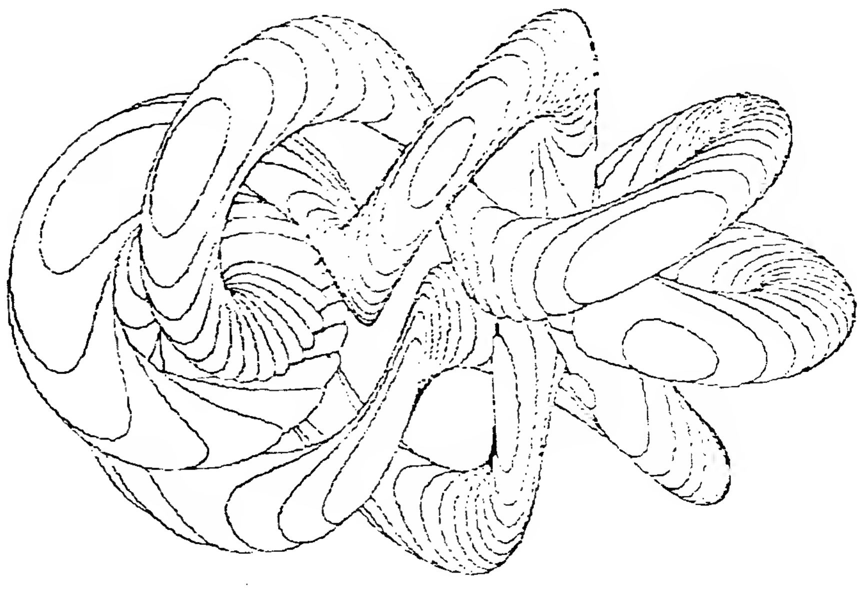
edm program1

to which the computer responds:

Segment not found.
Input:

Well of course it did not find that segment, because it never existed before.

BUT the computer had enough sense to assume that we wanted a new segment, so it set us up in INPUT mode, ready to type in the program. GO AHEAD.



EDIT

commands

Most of the commands which you can give the computer in the edit mode are abbreviated to one letter or character. USE ONLY THAT ONE CHARACTER, but remember them by their full name.

n you can move the pointer to the segment to the next line by this command. It will type out the next line after doing the task of moving the pointer. If you want to move down more than one line type in *n* followed by a space and the number of lines to be skipped. That is, *n 5* would move the pointer down 5 lines.

- To move up the segment, we use the minus sign (-). Like the next command (*n*) it will move the pointer and print out the line to which the pointer has now moved. You can type in *- 5* and the pointer will move up 5 lines (or however many you specify).

p Want to print out a line? Well set the pointer to that line and then just type in *p* . If you want to type out more than one line, then type in (say) *p 5* , and 5 lines, starting at the current line will be printed for you AND the pointer will be reset to the last line printed.

d Want to delete a line? Set the pointer to that line and then just type in *d* . Try the command *p* now. What happened? Try moving up or down from here. What happens? Can you delete 5 lines by typing in *d 5* ? What happens if you do a *p* , - , or a *n* now?

s THIS ONE IS IMPORTANT. IF YOU WANT TO MAKE A CORRECTION ON A LINE, MOVE THE POINTER TO THAT LINE, AND THEN TYPE IN

s/bad/good/

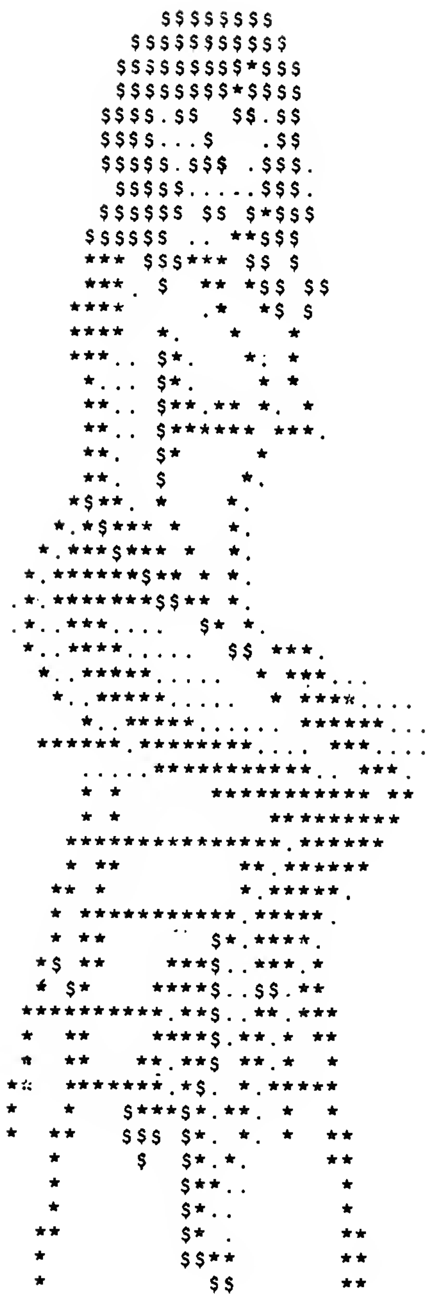
AND THEN MULTICS WILL SUBSTITUTE *good* IN THE LINE WHERE *bad* WAS BEFORE. BEWARE IF *bad* OCCURS MORE THAN ONCE, THEN EVERY INSTANCE OF *bad* WILL BE CHANGED.

w When you want to save the segment you have created for future use, then just type in *w* and the editor will put it away for you. If you make more corrections after saving a segment, then save again.

Putting a segment away requires the computer to write a copy into memory, so that's why *w* .

q When you are finished editing and creating the segment and are ready to do something else (like go home, or run the program), type in *q* for quit.

NOW YOU TRY THIS:



1. get into the edm editor and create the segment named program1 by the command:
   edm program1

2. input the handwritten program on the previous page. don't worry too much about correcting everything before you hit return.

3. once you've finished typing in the program, change mode (into EDIT).

4. move to the top of the segment (should be 11 lines long, so try - *11*. What happens if we type - *99* ?

5. using *n* and *s*, correct the program.

6. go back to the top and print out a clean copy.

7. repeat until you have got it right !!!

8. now lets make another type of correction. lets say that we need to insert the following line after the line " *do while* ..."
   / compute yearly balance */*

9. move the pointer to the " *do while* ..." line and then change mode to INPUT.

10. type in the new line, then return to edit mode. go up a couple of lines, print out (say) 6 lines, and make sure that you got the new line in the right place.

11. quit

12. you should have got a reminder at step 11, that you hadn't saved the segment. we hope you said *no* .

13. so now save the segment and then quit.

more EDM commands

l

using *n* and - to move around a segment can be pretty frustrating sometimes, so the locate command enables you to search the segment for a particular word, character or set of characters. For example, to locate the line containing *"do while ..."* in program1, you could have just typed *l do while* ! What would happen if the segment contains more than one *"do while"* or something else that you are searching for?

t

Want to get to the top of the segment? Well just type *t* . What happens if you now type in the command *p* ? Why?

b

Want to get to the bottom - try *b* . Notice anything? What mode are you now in?

=

Sometimes it is useful to keep track of a line by it's physical location in the segment. Invisible to you, Multics keeps a "line counter" with each line. You can get the value of this counter for the current position of the pointer by typing in =. If later you are at the top of the file use *n* and this value to get to that line.

move

You inserted a line at the wrong place right? Find out the line number of the line you want to move (=), put the pointer to the line after which the insertion is to be made, and then type *move 23* where 23 was the line number of the misplaced line.

k

If you are really brave, you can prevent the editor from verifying all the things you do (such as typing out the corrected line after a substitute) by killing the responses.

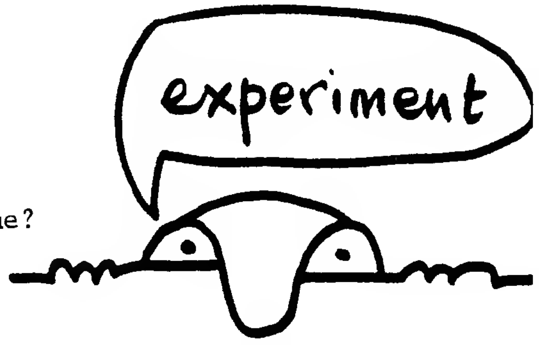
ON THE OTHER HAND,

v

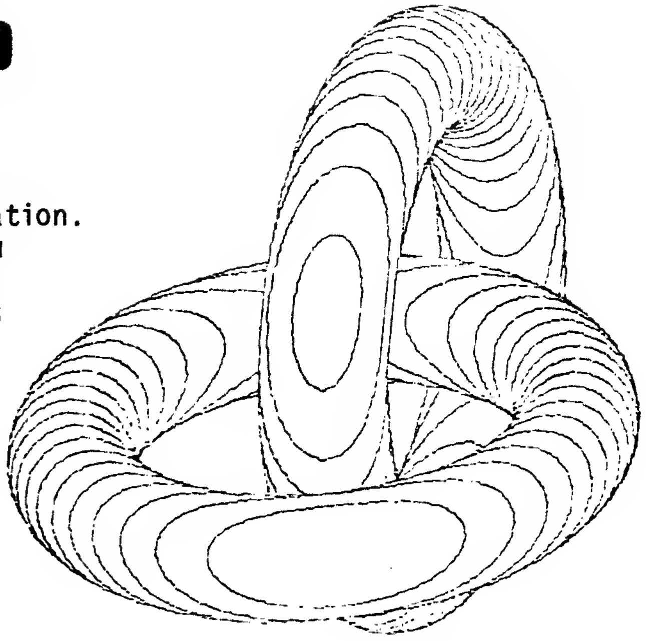
if you want the editor to be verbose, then type in *v* . The system is usually verbose.

,

Did you forget something at the end of a line? Try typing just a comma (,). Multics will type out the line, and then allow input!



HELP

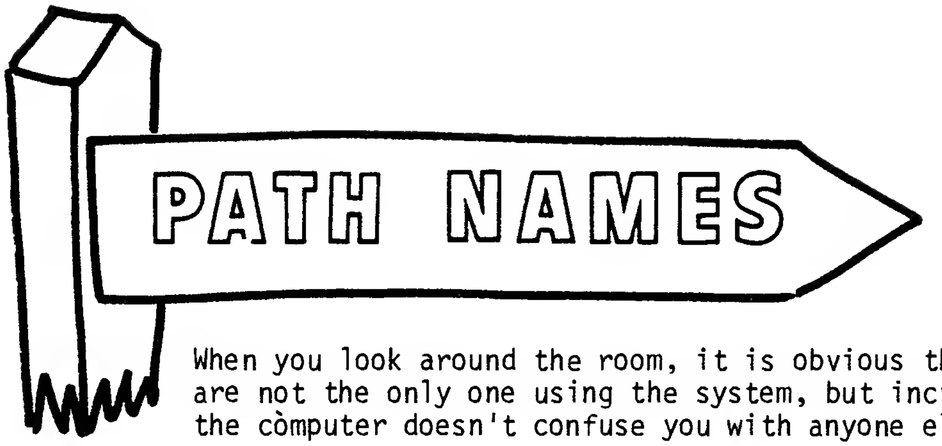


help is a Multics request for information.
BUT you have to tell it what you
want help about. Try typing
in *help help* which means
that you want help
about help !!

if you can't remember
the actual command that
you want to use but you
know that it starts with
(say) *q* then type in
*help q** and Multics
will tell you about
all the commands
that start with the
letter *q* (which may
take a long time !!)

LOGOUT

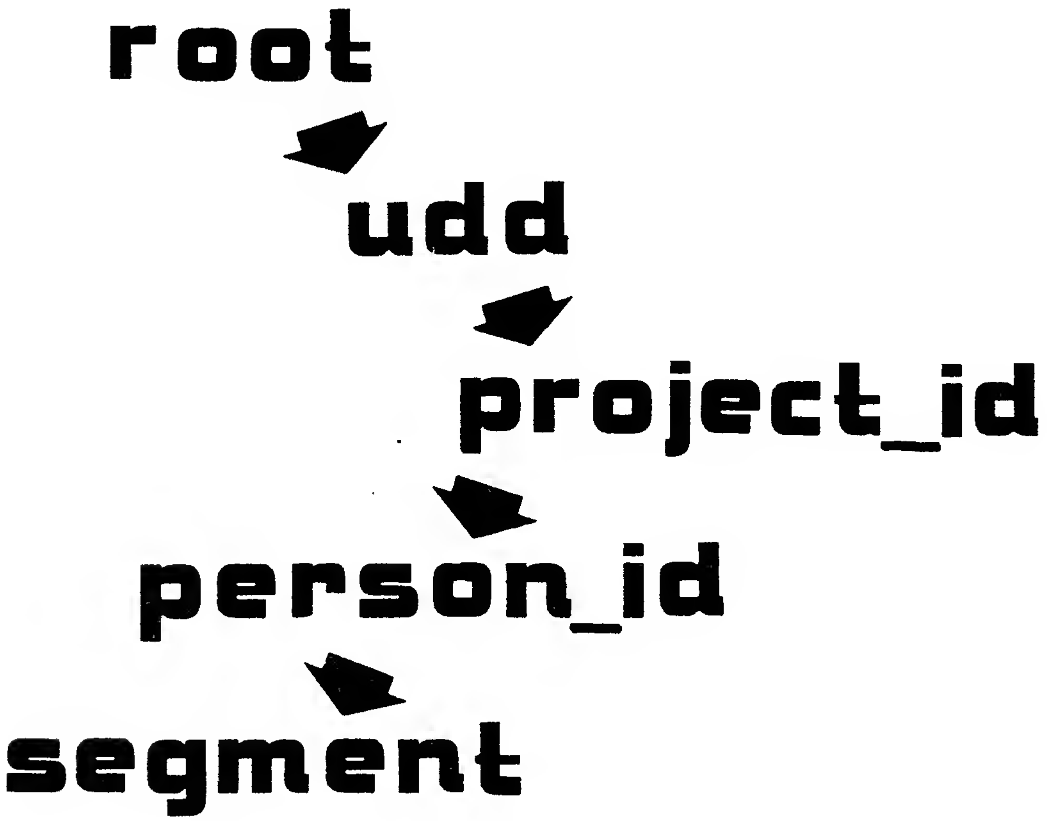
Once you get into the system, you also need to get off. It's not nice just to hang up the phone or to switch off the terminal, so be nice, just type in *logout* and the system will be much happier.



When you look around the room, it is obvious that you are not the only one using the system, but incredibly the computer doesn't confuse you with anyone else.

That's because the Multics system carefully files your data (segments) under a special place, just for you. Actually the computer thinks of you not as a person (heaven forbid) but as a DIRECTORY. In the directory which uses your name (person_id) is saved all the segments that you have created.

You exist (as far as Multics thinks) as a directory which is part of a project, and that project exists as a directory under something call *udd*, (user's directory of directories) which in turn exists under the "root" of the system. Your segments are known to you simply by their names, but if you wanted to get access to someone else's segments you would have to known the complete path from the root to that segment - called a *pathname*.



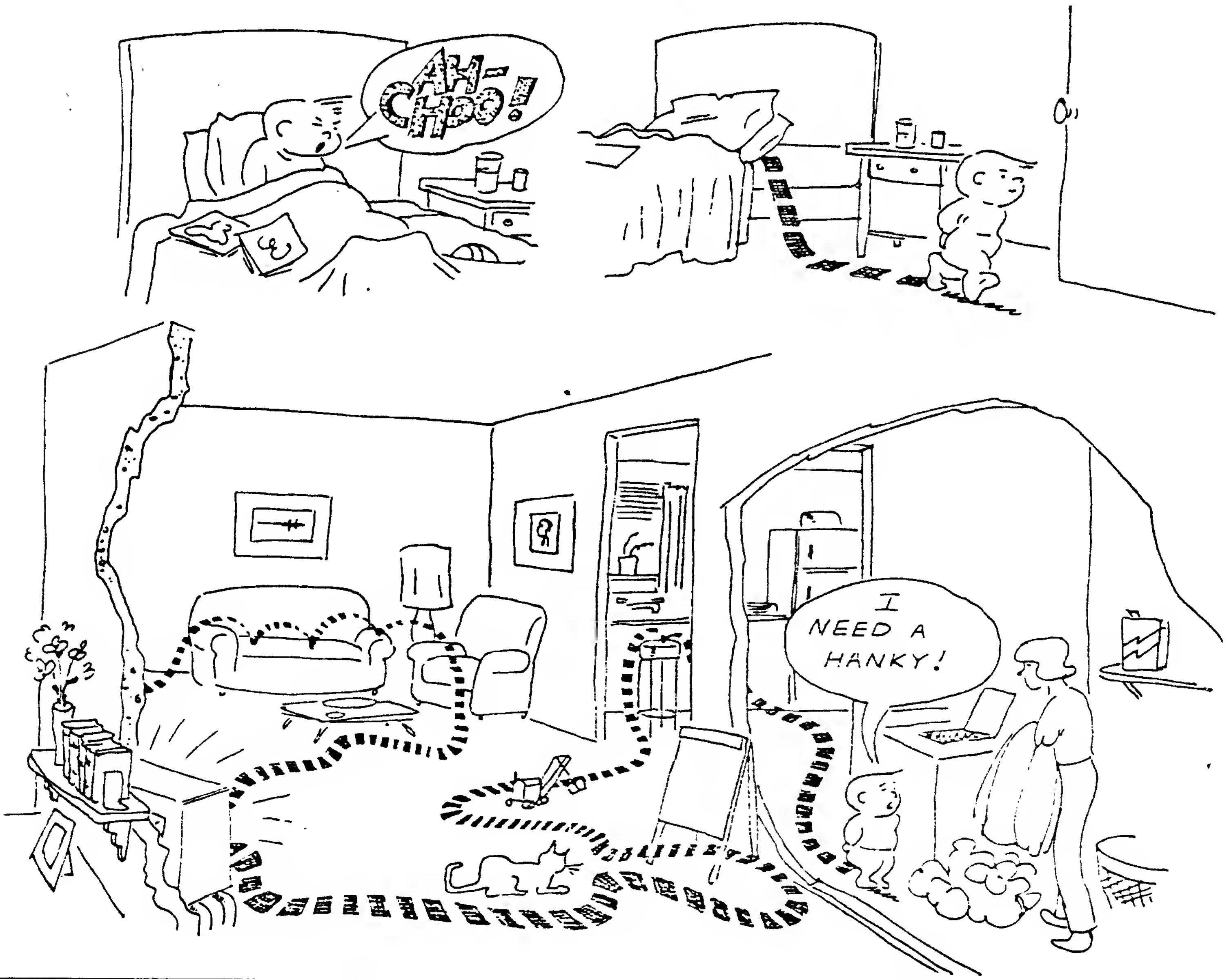
Instead of an arrow, Multics uses the > symbol, and since every path starts at the root, then the name "root" is omitted. The above path is named:

>udd>project_id>person_id>segment

NAMING SEGMENTS

The name you give
to a segment can be chosen
from almost anything you want to
make up, including letters, digits and
some special symbols (such as _ but not . or >).
If you are going to use a segment as a program source then
you must put a suffix on the name which is the name of the language
you are planning to use. A period separates the name and the suffix. So really
our program which we created earlier should be called *program1.pl1* but
we'll fix that later. Languages include, *pl1, fortran, basic* and
pascals

WARNING: Using some special symbols like @ # . * ? % can be hazardous to success!



Multics Commands

Almost everything you do in Multics involves giving commands to the system. *edm* is really a command to edit a named segment, just as *login* is a command to give you access to the system. There are literally <u>thousands</u> of commands; here are a few which you might find helpful.

Each command can be given in two forms - the long form (spelled out in something like English) and the abbreviation. We show the abbreviations here.

cp

Really means *copy*. You have to give it the name of the segment you want copied and the name that you want to save it as. This is generally where you need those long pathnames. The command

cp >udd>CS2980W79>LeeJAN>bubblesort sortprogram.pl1

would copy the segment *bubblesort* from person_id *LeeJAN* in project *CS2980W79* and save it in your directory as *sortprogram.pl1*

rn

Remember that we decided that our program named *program1* needed a suffix of *pl1* ? Well the rename command (*rn*) will do it for us:

rn program1 program1.pl1

ls

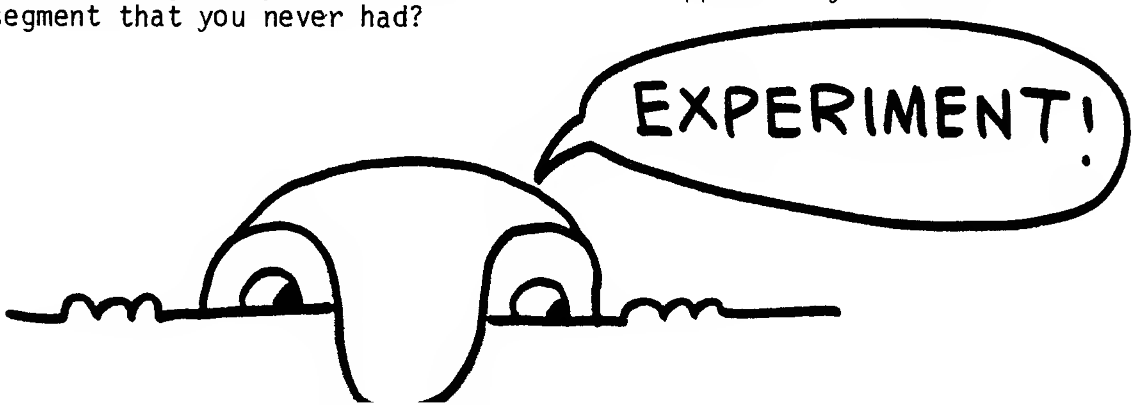
Have you been doing a lot of work and so have a lot of segments saved? Would you like to know what all their names are? Well the list (*ls*) command will tell us that, plus some more information which we won't worry about now.

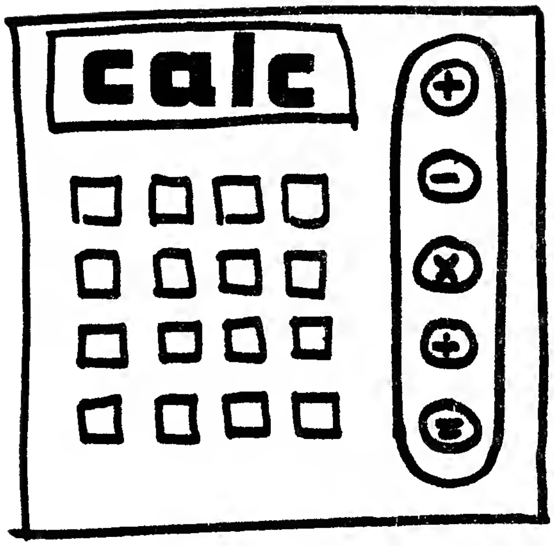
pr

Do you need to look at a copy of your segment on the terminal screen but don't want to go into edm to get it? Well try print (*pr*) followed by the name of the segment to be printed. It does a real nice job!

dl

There is going to come a time when you will have used up all the storage space that you are allocated. At that point you had better remove or delete (*dl*) some of your old segments. Don't forget to name the segment you want removed. What happens if you name a segment that you never had?





Did you leave your pocket calculator at home today - or did the battery run low? Well Multics will work like a calculator if you want without ever really having to write a program.

Just type *calc*

Apparently nothing happens - but now try *5+67-3* what did you get ? The system should have responded by giving you the answer to that calculation.

If your homework problem involves algebra, try something like the following:

```
a=5
b=10
c=56
d=a*2+b/3-c**4
```

What happened? Apparently nothing? Well, now try entering just *d* and you will get the answer:

```
=  -9834483
```

Once you have completed all your work quit the calculator mode by typing *q* (which means you had better not use *q* as a variable!!)

CHANGES

PASSWORD

When you get to the
stage where you would like
to change your password, you can
do the following:

NEXT TIME YOU LOG IN ENTER:

login person_id project_id -cpw

where *cpw* is an abbreviation for change password.

The system will respond with its usual output:

Password:

Enter your old password now; if it is OK then the system will ask for your new password twice just to make sure! Get it right both times! Thereafter your new password is in effect.

DEFAULT PROJECT_ID

When you filled in the request for a person_id, you also specified what your "default project_id" was to be. This is the project_id used at login if you don't specify a project_id. If you want to change this default situation, then do the following next time you log in:

login person_id project_id -cdp

where *cdp* means change default project_id.

Your default project_id will then be the one you have just logged in to.

This time the system won't ask you for your old default project_id nor will it ask you to repeat it

BUT

YOU MUST HAVE ACCESS RIGHTS ON THAT PROJECT BEFORE YOU CAN USE IT

COMPILING A PL/1 PROGRAM

ITS A CINCH

All you have to do is:

1) have prepared a segment in advance which contains the PL/1 source code,
2) have named that segment such that the segment name and the PL/1 procedure name are the same (with the exception of the suffix)*,
3) have given the segment name a suffix of pl1 (note the difference between the l(ell) and the 1(one) it is very important),
4) then simply type *pl1 program_name* (you don't need the suffix here).

THEN Multics does it's bit. It will compile your program from the source code segment and will prepare a new segment for you which is simply named with the unsuffixed name of the source segment...get that?

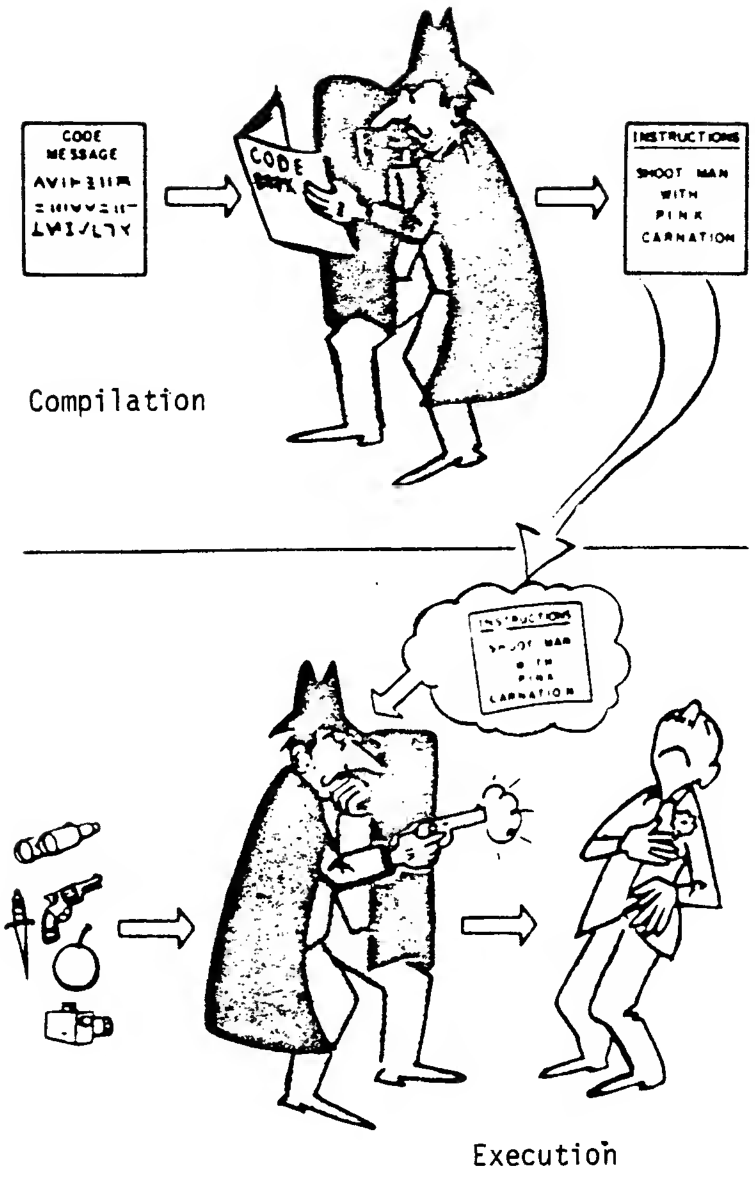
FOR EXAMPLE if the source code program had been contained in a segment named *catalog.pl1* then the produced code from the PL/1 compiler would be put into a segment named *catalog* .

OF COURSE

compiling a program isn't the same as running it ...

BUT ALL YOU HAVE TO DO NOW TO RUN YOUR PROGRAM IS JUST
TYPE IN THE NAME OF THE PROGRAM
AND IT DOES THE WORK.

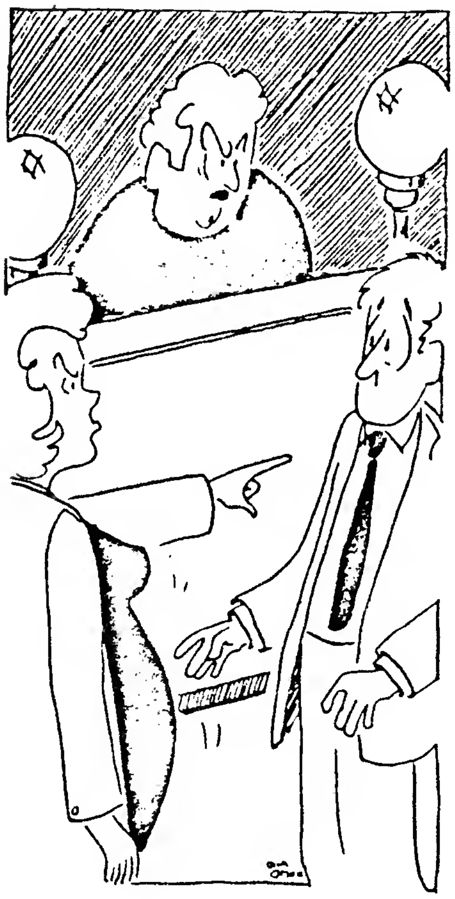
* Hopefully, you have tried to compile and run the segment named *program1* and have had a problem - this is why!!!



The two-pass nature of compilers—I. (From A. B. Kahn, "An Appreciation of Computer Appreciation," in *Proc. 22nd Nat. ACM Conf.*, Aug. 1967. Reproduced with permission of ACM and A. B. Kahn, Westinghouse Electric Corp., Baltimore, Md.)

WHAT DO YOU MEAN **IT DIDN'T** COMPILE?

There are two kinds of bugs in compilation -



THE WARNING

AND

THE

FATAL

ERROR

Most of the time, warnings don't hurt you because the compiler has made the correction for you ... but read the warning carefully to make sure it made the correct carrection.

BUT IF IT IS A FATAL ERROR go back and check your original program.

RUNNING A PL/1 PROGRAM

AS WE SAID EARLIER, RUNNING A PL/1
COMPILED PROGRAM IS EASY ALL
YOU HAVE TO DO IS TO
TYPE IN THE NAME
OF THE
PROGRAM!

NOTHING HAPPENED

if you used a simple GET or PUT in the program, the input and output
will be through the terminal at which you are sitting, and so
when it appears that nothing is happening it may be that
that is because _you_ haven't done anything like
giving the program some input data.

RECOMMENDATION: just before each input statement in your program, insert a PUT statement to tell you that the program needs input, such as

put list ("please input the data");

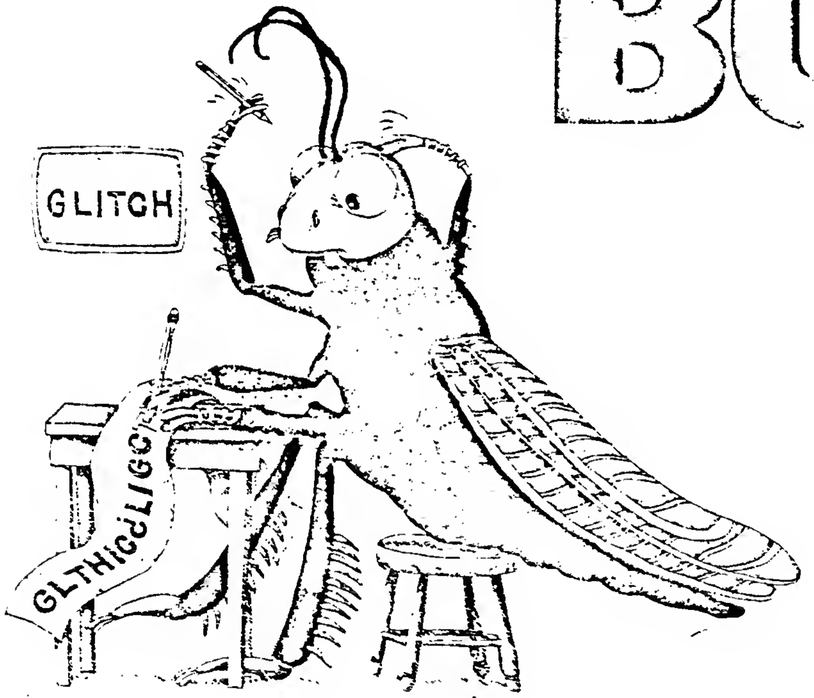
this we call a

PROMPT

DID YOU GET:

WRONG ANSWER! STILL NOTHING ERROR MESSAGE?

THEN YOU'VE PROBABLY GOT A **BUG!**



© Datamation

Did you know that according to Dr. Grace Hopper the original BUG was really a moth that got beaten to death in a relay in the Mark I computer at Harvard?

DO THE FOLLOWING - RECOMPILE THE PROGRAM USING THE COMMAND

```
pl1 program_name  -tb -map  -profile
```

where of course, you substitute your program name for *program_name*. Now we are ready to do some serious debugging.

DEBUGGING A PL/1 PROGRAM

The "things" after the program name which each start with a minus sign (-) are called control arguments. These tell the compiler to do some other things as well as just compiling the program. These will be useful during debugging.

tb creates a "symbol table" which is simply a listing of the symbols used in the program together with their location in memory.

map creates a special listing of the program as it compiled, containing the location of each statement in the compiled code, listings of all the symbols used and their locations (like *-tb* but for you to read) and saves all the error messages. This is a special segment named *program_name.list*. You can look at it by using the *pr* command.

profile at compile time, this argument makes the compiler put in some counters with each statement, so that later on we can look at how many times each statement has been executed - and that's very useful when you think that you have an infinite loop!

PROFILE

I input the program below which is named *ptest*.

After compiling and running it, I asked for its profile by just typing *profile ptest*. With a little bit of cut and paste, I've put the program, with its line numbers and the profile together.

```
ptest.list


   COMPILATION LISTING OF SEGMENT ptest
   Compiled by: Multics PL/I Compiler, Relea
   Compiled at: Virginia Tech Computing Cent
   Compiled on: 01/19/79  1540.4 est Fri
       Options: table map profile

1   ptest: procedure;
2     dcl i fixed decimal;
3      i = 1;
4      do while (i < 67);
5         if i < 30 then i = i + 2;
6         i = i + 1;
7         end;
8    end ptest;
```

```
profile ptest

LINE ST      COUNT


   3            1
   4            1
   4           47
   5           46
   5           10
   6           46
   7           46
   8            1
```

Notice that line 5 has two counts - one for the test and one for the "then part". Notice also that the do while statement is executed one more time than the rest of the loop - obviously!

The profile which was printed out also had a column headed "COST" - don't worry about that one yet. Concentrate on the COUNT.

QUESTION: why do you think that line number 4 appears twice in the count?

PROBE

probe is a special processor in the Multics system which allows the user to
examine the results of partially executing a program. However,
only certain programs written in certain languages
can be "probed". PL/1 is one of those
languages and a PL/1 program
which has been compiled
with the *-tb* option
satisfies the
other
condition.

After the program execution has been halted either by a breakpoint (see below) or by the use of the break key, enter:

probe program_name

a Once into the probe system, you can set an automatic stopping point (called a "breakpoint") after any statement by typing *a N* where *N* is the line number of the chosen statement. Then when you run the program again, it will automatically stop and enter probe.

b If you wanted to set a breakpoint <u>before</u> a statement, type *b N* You can set several breakpoints in a program.

r N will reset a breakpoint - more about that later.

c If you want to continue running the program, either after setting breakpoints or examining the program status, enter *c*

V After the program has halted, it is very useful to examine the values associated with the variable identifiers. For example, if your program contained the identifier *total*, then the probe command *v total* would print out the current value for you. If *total* is an array then probe will output all the associated values.

l Sometimes during the execution of a program, you want to change the value associated with a variable identifier. This can be done using *l* for let; *l total = 56* would reset the value associated with *total*; *l total = total - 1* will decrement *total*.

BREAK

on each
terminal there
is a button marked
BREAK

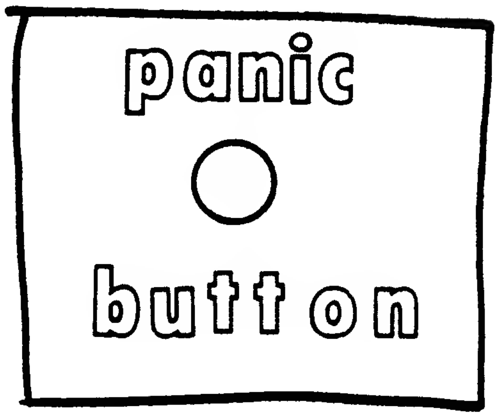
When you press this button smartly, it will cause the system to stop whatever it is currently doing and to permit you to tell it what to do next.

The running operation is actually "suspended" so that you can resume the operation at any time by typing
start

IF YOU ARE NOT CAREFUL, YOU MAY HAVE SEVERAL PROCESSES SUSPENDED AT ONE TIME - IF YOU DONT PLAN TO RESUME THEM ENTER *rl -all* ;* THE READY MESSAGE TELLS YOU HOW MANY YOU HAVE SUSPENDED - IT IS CALLED THE LEVEL NUMBER.

If you are using the probe deugging aid, a suspended process is restarted by the probe command *c(*ontinue*)*.

* *If you couldn't work it out*
rl is short for release



NOW YOU TRY THIS:

Compile the following program with the options *-tb -map -profile*

```
mortgage: procedure;
 declare (c,p,r) float decimal,
         (n,i) fixed decimal;
     get list (c,p,r,n);
     i = 1;
     do while (i <= n);
        c = c*(1.0e0 + p*0.01e0) - r;
        put skip list("after year",i,"  balance is", c);
        i = i + 1;
     end;
 end mortgage;
```

This is what happened:

```
pl1 mortgage -tb -map -profile
PL/I

WARNING 75
The undeclared identifier "sysprint" has been
contextually declared as a file constant. It will
acquire default attributes.

WARNING 75
"sysin"
```

Note that since the same warning occurs twice, the description of what it means is not repeated - only the number is repeated.

As its name implies this program computes the principal remaining in a mortgage (c) with a yearly payment of r with a interest percentage of p, over a period of n years.

BEFORE RUNNING THE PROGRAM *mortgage* SET A BREAKPOINT AFTER THE LINE NUMBERED 7 BY:

```
probe mortage
a 7
```

the probe processor will then reply:

```
Break set after line 7 of mortgage.
```

At this point, we have never run mortgage so we cannot tell the system to continue... so instead we quit probe

```
q
```

and begin to execute mortgage:

```
mortgage
10000
12
375
10
```

after the last input, the system halts after typing:

```
Stopped  after line 7 of mortgage.
```

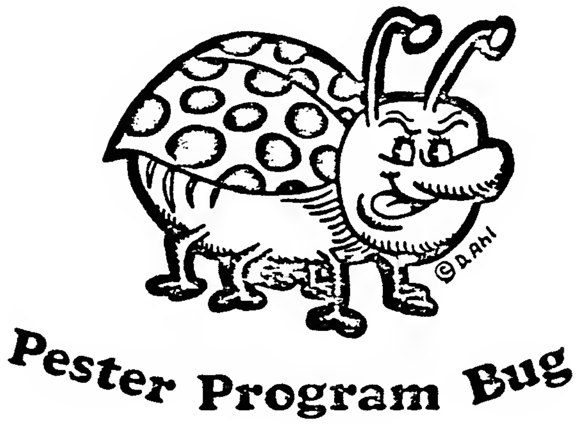
The rest of the dialog went like this:

```
v c
   10825
v p
   12
c
after year   1   balance is  1.0825000e+004 Stopped after line 7 ...
```

We repeated this process several times until we were tired of the task, and wanted the program to continue freely. We found that the next time the breakpoint was reached we typed in
r and Multics responded with *Break reset after line 7 of mortgage.*
Then when we entered *c* the program ran to the end without stopping again.

TWO NEW PROBE COMMANDS

r	reset the breakpoint
q	quit



ELECTRONIC MAIL

IN 1964 WHEN DARTMOUTH COLLEGE FIRST INAUGURATED THEIR TIME-SHARING SYSTEM, TWO OTHER COLLEGES ALSO GOT TERMINALS INTO ■DTSS■ - SMITH COLLEGE AND MOUNT HOLYOKE - AND SO WAS STARTED THE FIRST INTER-COLLEGE ELECTRONIC MAIL SYSTEM. IN FACT THERE ARE AT LEAST TWO MARRIAGES WHICH ARE ATTRIBUTED TO BLIND DATES WHICH WERE DEVELOPED USING DTSS
■ AND PROBABLY LOTS OF OTHER THINGS ■

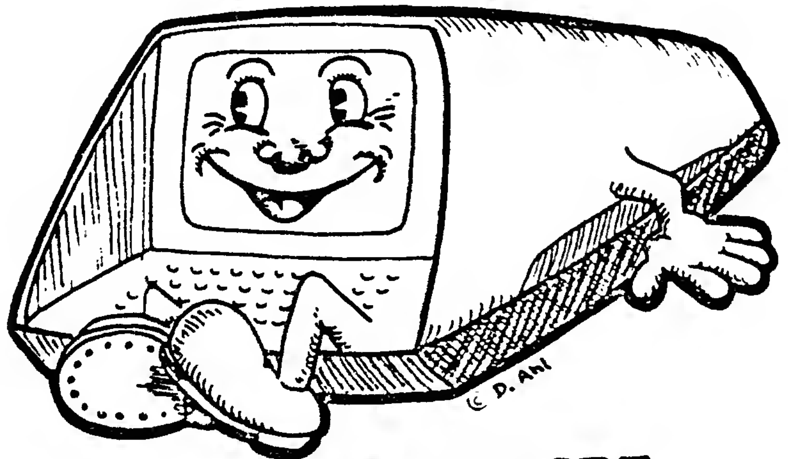
you can communicate with other persons on the Multics system through the mail system. some faculty members even use the mail system to permit their students to submit their homework assignments electronically!

USE HELP TO FIND OUT MORE ABOUT MAIL AND SEND_MESSAGE

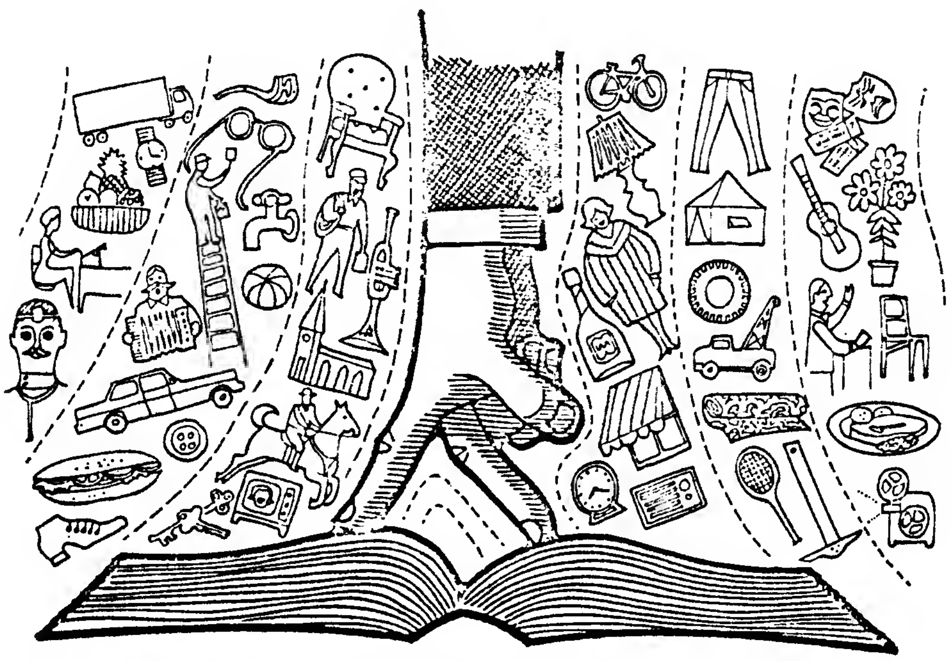
Remember *help?*
If you didn't then you need to enter

help mail or
help send_message

to get this information.



Cyrus CRT



Let your fingers do the walking...

Ⓒ *C&P Tel. Co.*

THROUGH THE DIRECTORIES ....

There comes a time in the life of every "Multician" when you want to do things like -

BORROW A SEGMENT FROM SOMEONE ELSE

GIVE A SEGMENT TO SOMEONE ELSE

GET ACCESS TO A SYSTEM FILE

two things you need to know -

the path_name of the segment you want to get to

the access rights of that segment

Let's say that you are the borrowee rather than the borrower, then there are two things you have to do:

1) tell the borrower the path_name to your segment, or if you are going to let him/her get everything, then the path_name to your directory (usually *>udd>project_id>person_id*)
2) set the access rights on that segment or directory to include the borrower.

ACCESS to SEGMENTS

When you want to refer to a borrower to give access rights, the path_name of that person gets reversed and is written as:

*person_id.project_id.**

where the star indicates any process class - just use * for the time being. If you wanted to give *WesselkamperTC* access to a segment irrespective of which project they were logged on to, you could use the name

WesselkamperTC..**

sa the set_access command needs to know three things:

1) the segment you are setting access to,
2) the rights you are giving (read, execute or write),
3) the name of the person to whom you are giving these rights.

SO IF WE WERE TO GIVE WESSELKAMPER THE RIGHTS OF READING AND WRITING TO THE SEGMENT NAMED ■ANYTHING■ WE WOULD ENTER:

sa anything rw WesselkamperTC..**

where

r means to give READ access (which includes the right to copy)

e means that the borrower can EXECUTE this segment

w means that the person named can WRITE into that segment, which includes the right to delete the segment altogether!!!

If you want to find out who has access to your segment enter

la segment_name ← **la**

and then if you want to delete someone's access rights try:

*da segment_name person_id.project_id.** ← **da**

Who the heck is SYSDAEMON who keeps appearing when you do an *la?*

There are lots of daemons running around in Multics - they do all the work!!

ACCESS to DIRECTORIES

You can give someone access to your directory (the list of segments that you have created) with or without the ability to get more detailed access to the segments in that directory. The command to set access is still *sa* but the path name is now the name of a directory. If it is to be your regular, working directory, you can just use the abbreviation *-wd* but if it is some other special directory, then you know much more than we are assuming at this point! Name your own directory!!!

If you want to know the name of your directory enter *pwd* (print working directory).

To give WesselkamperTC full rights to my directory, including the rights to set his own access rights to all the segments in there and the ability to add new segments (called appending) we would enter:

sa -wd sma WesselkamperTC..**

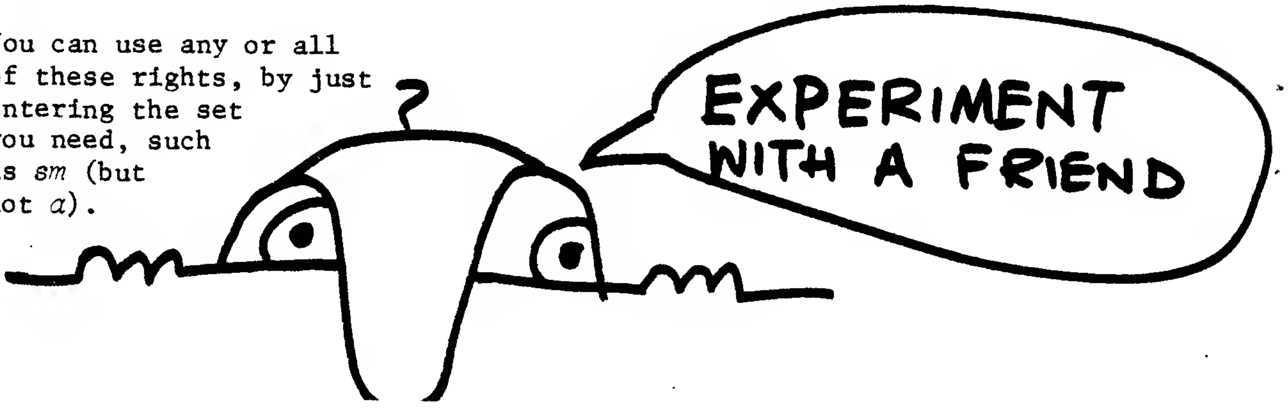
where

S means that we gave him "status" rights - he can enquire of his access rights to a segment and list the contents of the directory.

M means that we permit him to "modify" the access rights of <u>any</u> segment in the directory (which gives him a pretty free access)

a means that we permit him to "append" segments to the directory. *Webster says: "append: ... to add as a supplement..."*

You can use any or all of these rights, by just entering the set you need, such as *sm* (but not *a*).



NOW YOU TRY THIS:

With a friend (or colleague):

send him a message giving him the name of one of your segments and suggest that he try to copy it to his directory. Tell him to let you know what happens.

He should get a message from Multics that says he doesn't have the correct access rights.

Give him read access and tell him to try again.

This time he should not have any problem.

Give him status access to your directory and ask him which segment he would like to see. Set the rights and let him have it.

NOW EXCHANGE POSITIONS AND SEE IF YOU CAN GET TO ONE OF HIS SEGMENTS AND TO HIS DIRECTORY. ONCE YOU GET A SEGMENT OVER TO YOU WHAT ACCESS RIGHTS COME WITH IT ?

When you are satisfied that you have a good knowledge of what is happening, clear up all the accesses to what they previously were.

A NOTE:

At Virginia Tech, we all live by the honor code; faculty leave doors open, students leave books on a shelf and we don't expect anyone to touch these things. So it should be with people's segments - we look at them when we are told explicitly that we can - and the same goes for people's directories.

Just because someone doesn't explicitly set the access to exclude you doesn't mean you have permission to look!

ABBREViate

if you are the kind of typist that I am, it is very rare that you can type a line without a typing error. although the use of # can help, sometimes it is easier to do less typing.

the ABBREV system lets you define some abbreviations especially for you which will assist you in your typing. for example, my project_id is CS2980W78 which I abbreviate to proj.

abbrev

this command initiates the abbreviating system, allows you to specify new abbreviations and to recognize old ones. once defined, abbreviations are saved from session to session, all you have to do to use them is to invoke *abbrev*

.a

all abbrev commands start with a period (.)
.a abbrev_name long_name specifies that from here on when you type *abbrev_name* you mean *long_name*

.l

if you forget what your abbreviations are, then *.l* will list them all for you.

.q

there are times when you don't want your abbreviations to be used; *.q* quits using abbreviations. *abbrev* restarts the use of abbreviations.

.d

you want to delete an abbreviation? Try this ... but don't forget to tell it which abbreviation to delete ... can you work out how?

Two useful abbreviations are *me* for your person_id (after you have logged in) and *proj* for your project_id. Others might include the person_ids of your friends if you want to communicate on the system.

STARS & STRIPES

(that is, the double bar or equal)

Many times you want to be purposely vague about a segment or segments name(s) because either you want to talk about ALL segments with certain styles of names or you don't know what the actual names have been stored as.

***** can be used in a segment_name to mean "any set of characters". For example, if I want to list the set of segments in my directory which have the suffix *pl1* we can enter:

```
ls *.pl1
```

and we will get a listing of only those segments which have that form. If we wanted to get all those segments which have double-barrelled names we could use

```
ls *.*
```

****** has a wider meaning that a single star. A single star means "any set of characters" except certain special characters, such as period (.) . So ** has the meaning of ALL irrespective of delimiting periods. For example, if we wanted to copy all the segments from user *Stu22* which have a suffix of *pl1* irrespective of their "first names", without changing the names. Even if the first names are single, double or triple barrelled (or more) we want the segments:

```
cp  >udd>CS2980W79>Stu22>**.pl1
```

= there are times when you don't want to repeat yourself. So in a single command we can tell Mutlics to use the same name over again by the = symbol. For example, lets rename a segment so as to add the suffix *pascals* :

```
rn  testprog  =.pascals
```

which is the same as having entered:

```
rn testprog  testprog.pascals
```

NOTE:
The stars and bars don't always work with all commands, but don't worry, if Multics can't do it, it will tell you!